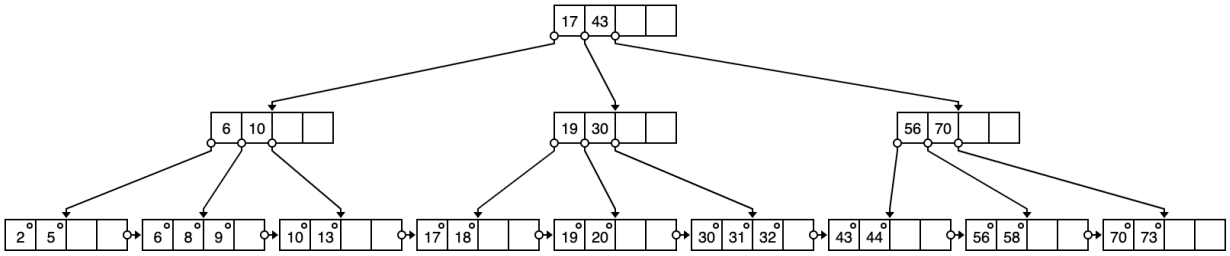
Madeleine Thompson

DSCI 551 Spring 2021

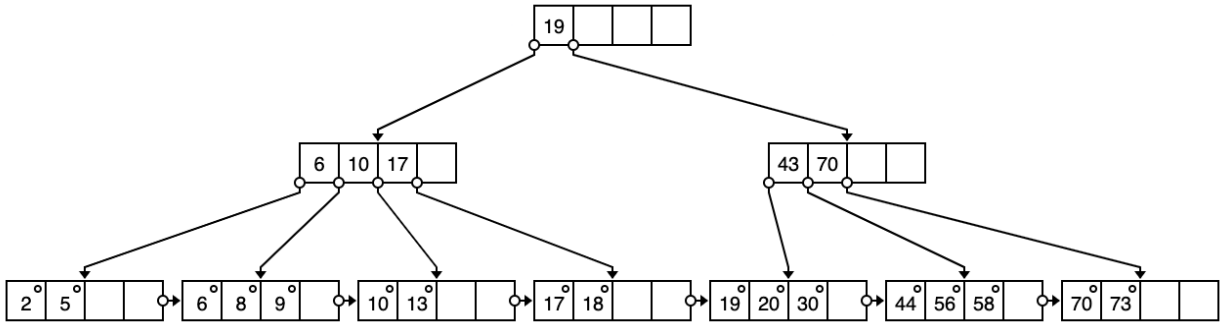
HW 4 – Indexing and Query Execution

1. (a) You would first follow the query "age >= 10". Read the root node, follow the root node to the left, read that node, then follow the pointer for >= 10 and read that leaf node. Then, read the last key in that node (13) and check if that is <= 20. It is, so continue on to the next sibling node to the right, read that node, and check if the last key in that node is <= 20. It is, so follow the pointer to the next sibling node and read that node. Stop at that leaf node because it contains 20. **In total, 5 block I/O's.**

(b)



(c)



2. (a) For a block-based nested-loop join with R as the outer relation, you would read R once and then run through R and read S once for each run of R for a total cost of B(R) + B(R)*B(S)/(M-2). There will be 2,000/(102-2) = 20 runs of R of size 100 blocks each, and there will be 50,000/(102-2) = 500 runs of S of size 100 blocks each. Thus, there will be one run through R and 500 runs through S, and the total cost of the nested-loop join with R as the outer relation will be:

2,000 + (2,000*50,000)/(102-2) = **1,002,000 I/O's**

(b) For a sort-merge join, you will first split R into 20 runs of size 100 blocks each, and then split S into 500 runs of size 100 blocks each since you are only using 100 pages for sorting. Then you will merge 101 – 1 = 100 runs from R and S and output a tuple on a case-by case basis. This would result in an overall cost of 3B(R) + 3B(S). However, the sort-merge join assumes that the buffer is enough to hold join tuples for at least one relation. Since S is a large relation and has 50,000 blocks and B(R) + B(S) = 52,000 > 100^2, you will have to do further merging because the join cannot be done by using only a single merging pass. This means that you will have to load S into memory and merge S to reduce its size before merging with R, which will add a read and write of S, bringing the total cost of this sort-merge join to 3B(R) + 5B(S). Thus, the total cost is:

3(2,000) + 5(50,000) = **256,000 I/O's**

(c) For a partitioned hash join of R and S, you will first hash S into 101 – 1 = 100 buckets and send all buckets to disk (1 load and 1 write of every block of S). Then, you will hash R into 101 – 1 = 100 buckets and send all buckets to disk (1 load and 1 write of every block of R). Then, join every pair of corresponding buckets. The cost of the partitioned hash join will be 3B(R) + 3B(S), assuming that min(B(R), B(S))/(M-1) <= M-2 since we are not using a hash table to speed up the lookup.

First, check that min(2,000, 50,000)/(101-1) <= (101-2), which is true because 20 <= 99.

Next, calculate the total cost of using a partitioned hash join of R and S:

3(2,000) + 3(50,000) = **156,000 I/O's**

Thus, **the partitioned hash join of R and S is the most efficient join algorithm**.

(d) For an index join of R and S where R and S are both clustered and S.a is a clustered index, you will iterate over R once completely, and for every tuple in R, fetch all of the corresponding tuples from S. Here, you have to assume that S has an index on the join attribute, and it is, S has a clustered index on the join attribute a. Because the index is clustered, you will calculate the total cost as B(R) + T(R)*B(S)/V(S,a).

The total cost of using an index join of R and S:

2,000 + 10,000*50,000/5 = **100,002,000 I/O's**

When comparing an index join and a block-based nested loop join on this data, NLJ will win because the value of V(S,a) is too small to make the index join more efficient.